Microsoft

A SSLippery Slope

Unraveling the Hidden Dangers of Certificate Misuse

Bill Demirkapi

Who am I?

- Security Researcher for the Microsoft Security Response Center
- Undergraduate at the Rochester Institute of Technology
- Low-level background in reverse engineering, exploit development, and OS internals
- Defensive and Offensive security background
-  Windows Internals

Agenda

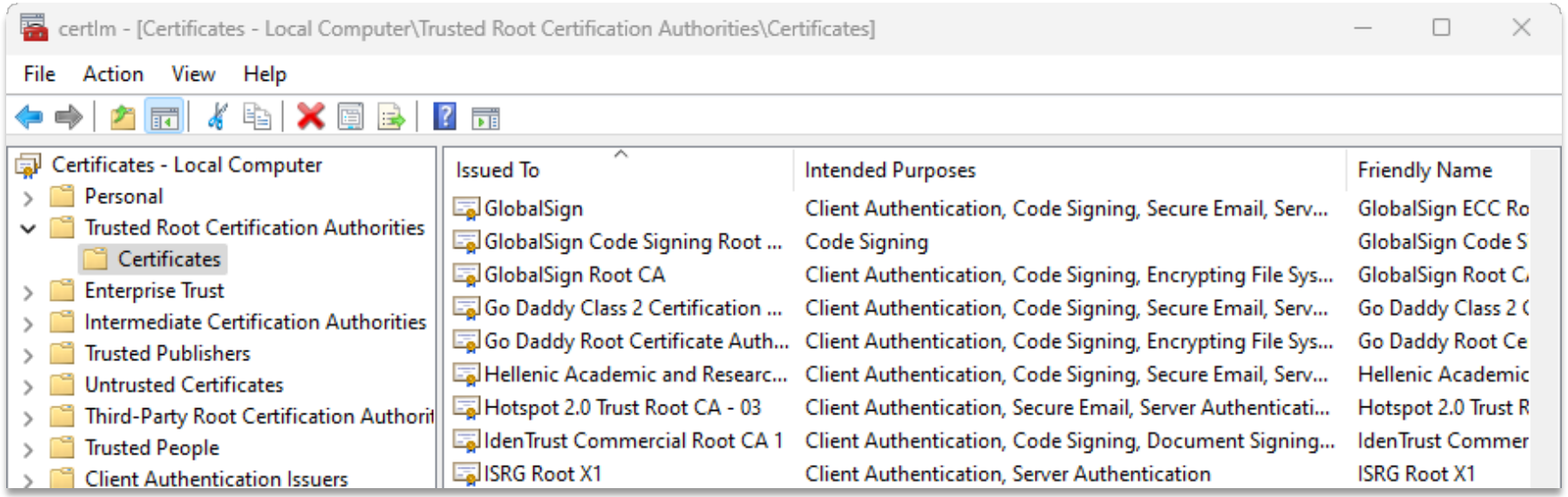
- How Are Digital Signatures Validated?
- Attacking Digital Signatures
- Abusing the Lack of [redacted] Validation
- Impact on Authenticode Implementations
- Impact on [redacted] Implementations
- Impact on Everything Else
- Review & Conclusion

How Are Digital Signatures Validated?

Digital signature verification in the real-world.

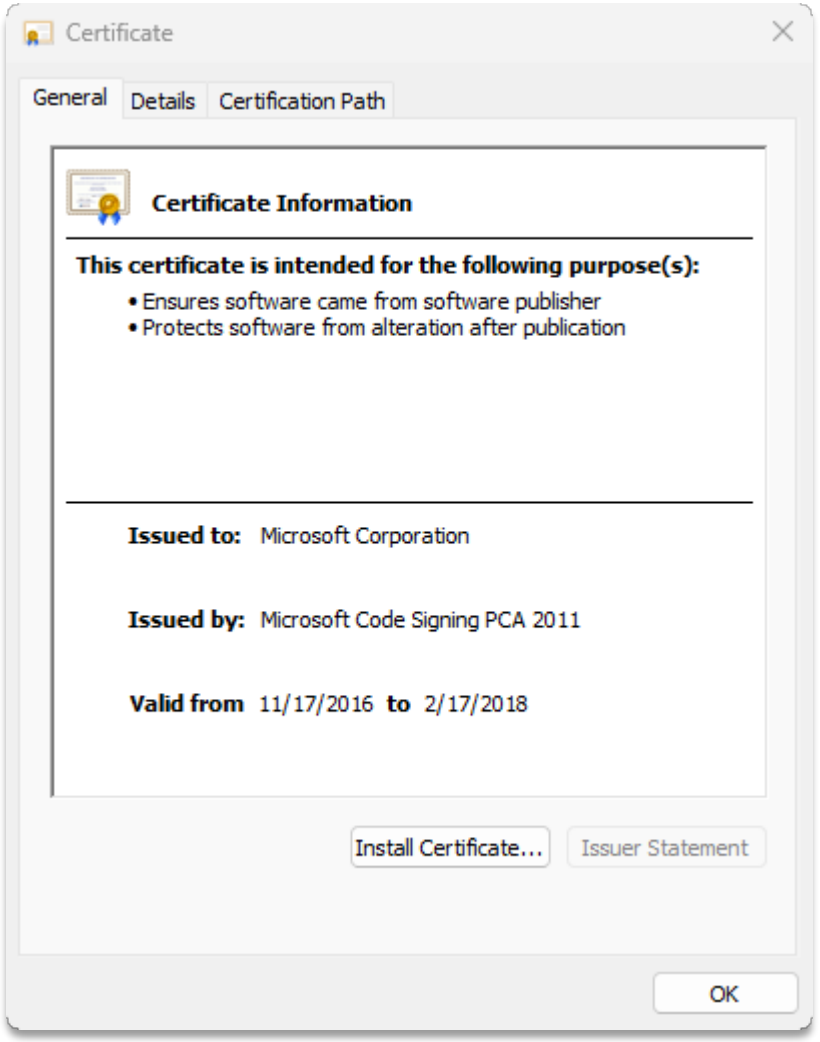
The Role of Trusted Root Certificate Authorities

- Trusted Root Certificate Authorities (CAs) are entities that issue "digital certificates".
- Root CAs are responsible for verifying the identity of entities (e.g., companies, individuals) before issuing them a certificate.
- Who verifies the root CAs? They come pre-installed!



Digital Certificates

A digital **certificate** is a digital document issued by a trusted third party, known as a Certificate Authority (CA).



- To validate a digital signature, the recipient needs to have the sender's public key.
- If the sender simply sent their public key along with the message, it could be modified in transit.
- Digital certificates contain the sender’s public key and are signed by a CA.

Digital Certificates: Chain of Trust

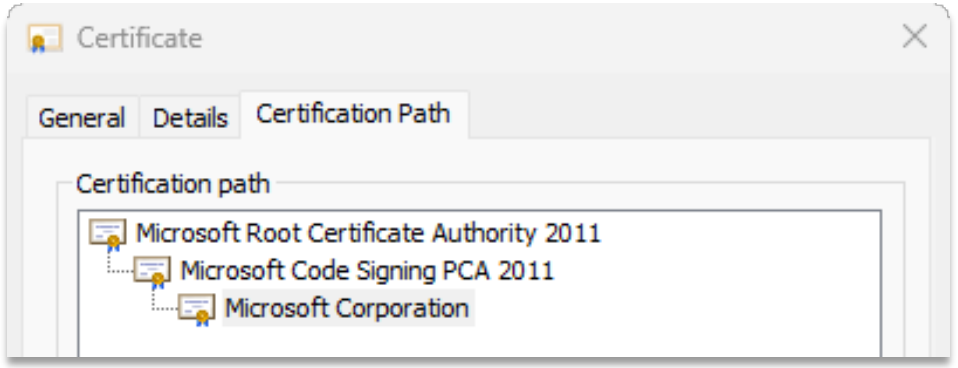
How are digital certificates verified? Introducing the Chain of Trust!

Root Certificate Authorities verify…

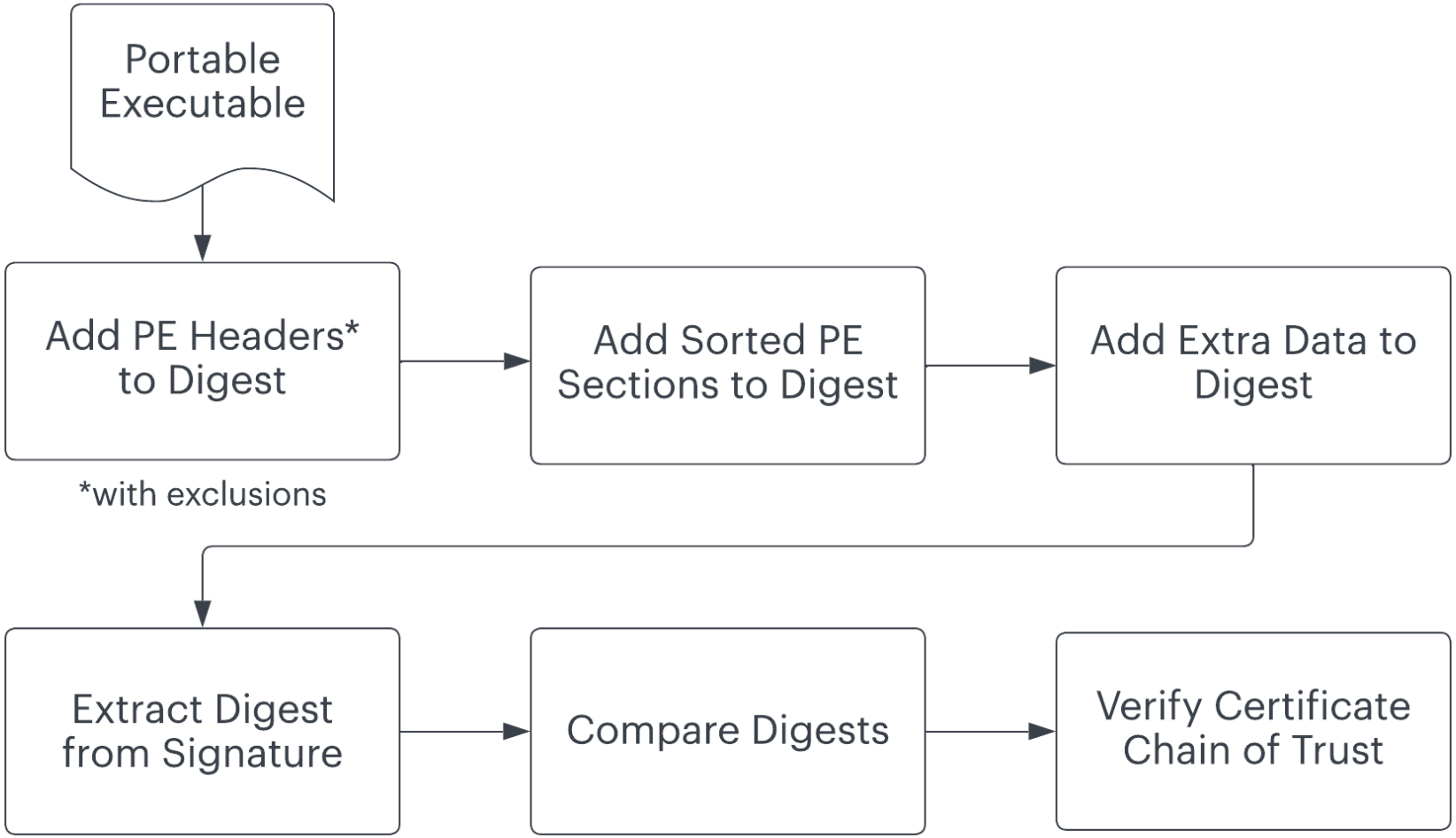
Intermediate CAs, which verify…

End-entity or server certificates, which verify…

A digital message!



Example: Microsoft Authenticode (Simplified)



Attacking Digital Signatures

What do we look for? Where do we start?

History

Digital signatures have been subjected to numerous attacks. Let's review a few notable examples!

- **1996**: Hans Dobbertin publishes a paper describing MD5 collisions and discouraging its future use (used for digests).
- **2004**: First publication of an MD5 collision.
- **2008**: Rogue CA is created with an MD5 collision.
- **2011**: The DigiNotar CA is compromised by attackers.
- **2012**: The "Flame Malware" abuses an MD5 collision to create a fake Microsoft CA.

Different Types of Attacks

Let’s discuss the attacks that are relevant for digital signatures:

- Memory corruption
  - How can we find them? Manual or guided analysis (e.g., CodeQL), fuzzing, etc.
  - How can we prevent them? Minimize attack surface, “safe™” languages, mitigations, “shift left security”, etc.
- Logic flaws
  - How can we find them? Manual or guided analysis (e.g., CodeQL), fuzzing, etc.
  - How can we prevent them? Minimize attack surface, “shift left security”, security-aware developers, automated testing of known attacks, etc.
- Cryptographic flaws
  - How can we find them? Mathematical analysis, algorithm audits, etc.
  - How can we prevent them? Use well-vetted algorithms. Don’t roll your own crypto.

Why Implementation Flaws Will Always Exist

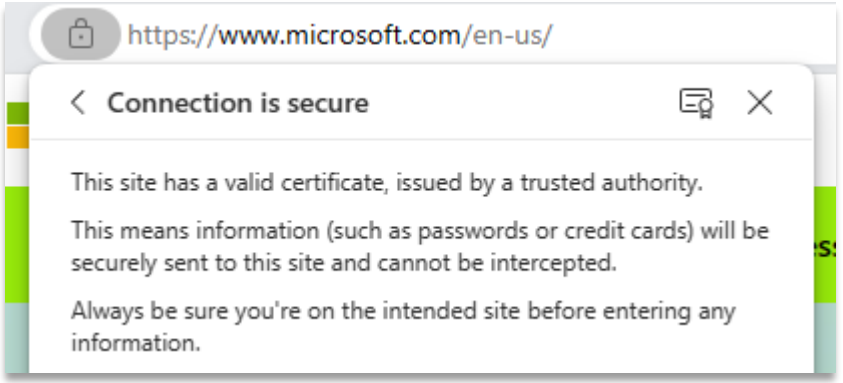
Logic flaws are one of the harder types of vulnerabilities to address.

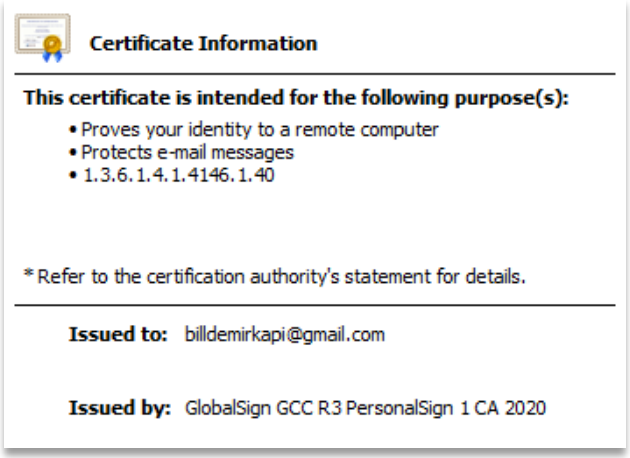
- Writing cryptographic systems is challenging due to their inherent complexity.
- Cryptographic implementations aren't isolated; they interact with a multitude of other systems, protocols, and software.
- Patching implementation issues can get complicated.
  - One reason why XYZ insecure "feature" (e.g., MD5) is difficult to get rid is because if you simply prevented its use entirely, you may break legacy systems and applications.
- Memory corruption is better understood. Logic flaws are harder to prevent, as they are context-specific.

Types of Signing Certificates

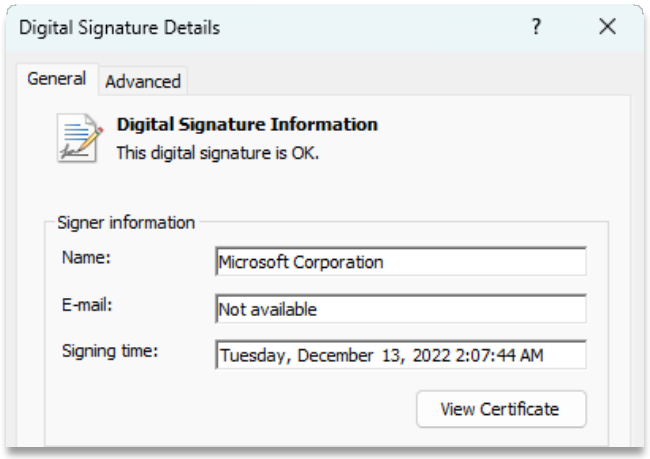
What are common types of certificates that CAs will issue today?

- SSL/TLS Certificates
  - **Purpose**: Secure connections between web servers and clients.
  - **Validated Information**: Domain ownership.
  - **Limitations**: Only intended use is for securing network connections.
- S/MIME Certificates
  - **Purpose**: Secure email communications.
  - **Validated Information**: Email account ownership, sometimes the identity of the individual or organization.
  - **Limitations**: Only intended use is for email security.





Types of Signing Certificates





- Code Signing Certificates
  - **Purpose**: Ensure the integrity of software and prevent unauthorized tampering.
  - **Validated Information**: Identity of software developer or organization.
  - **Limitations**: Used strictly for signing software binaries, scripts, and code.
- Document Signing Certificates
  - **Purpose**: Confirm the authenticity and integrity of documents.
  - **Validated Information**: Identity of document signer.
  - **Limitations**: Only intended use is for digital signatures on documents.

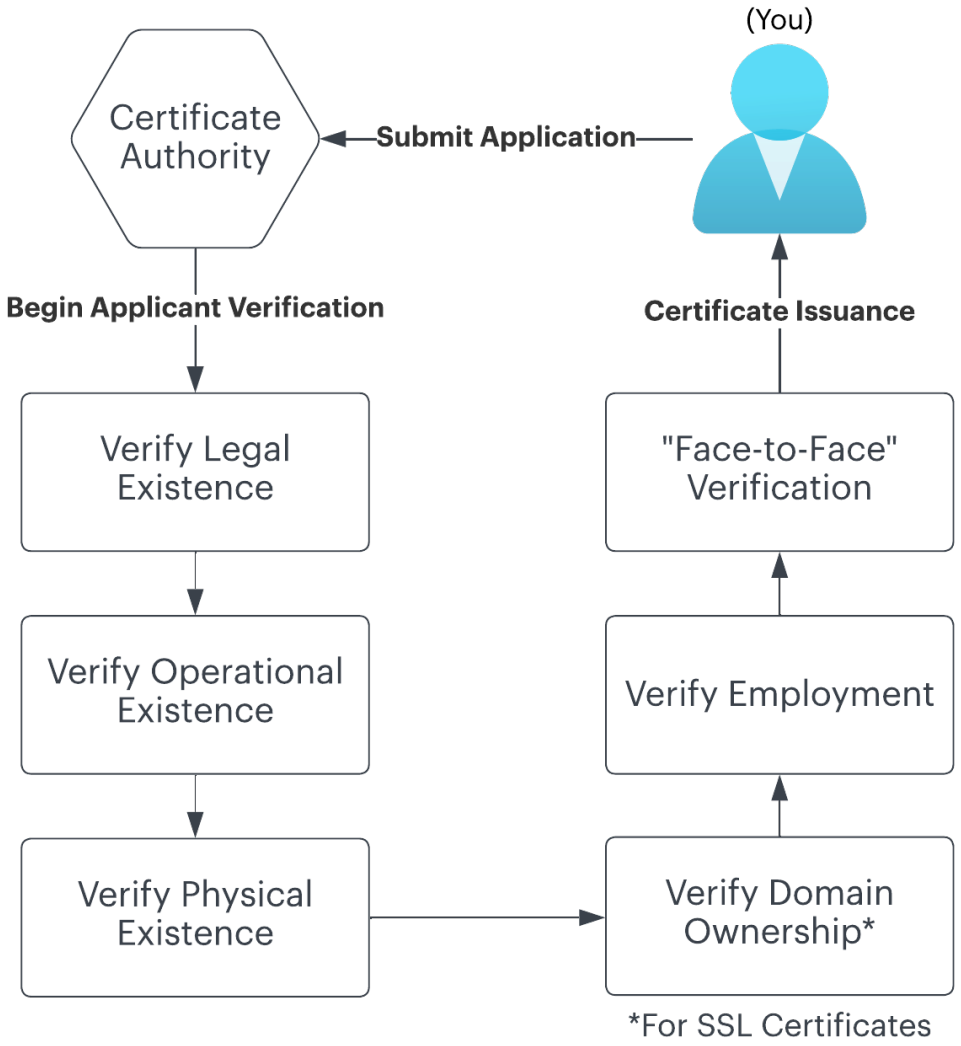
Comparison of Identity Verification Requirements

Not all certificates are made equal. Depending on their intended usage, the identity verification requirements can vary greatly.

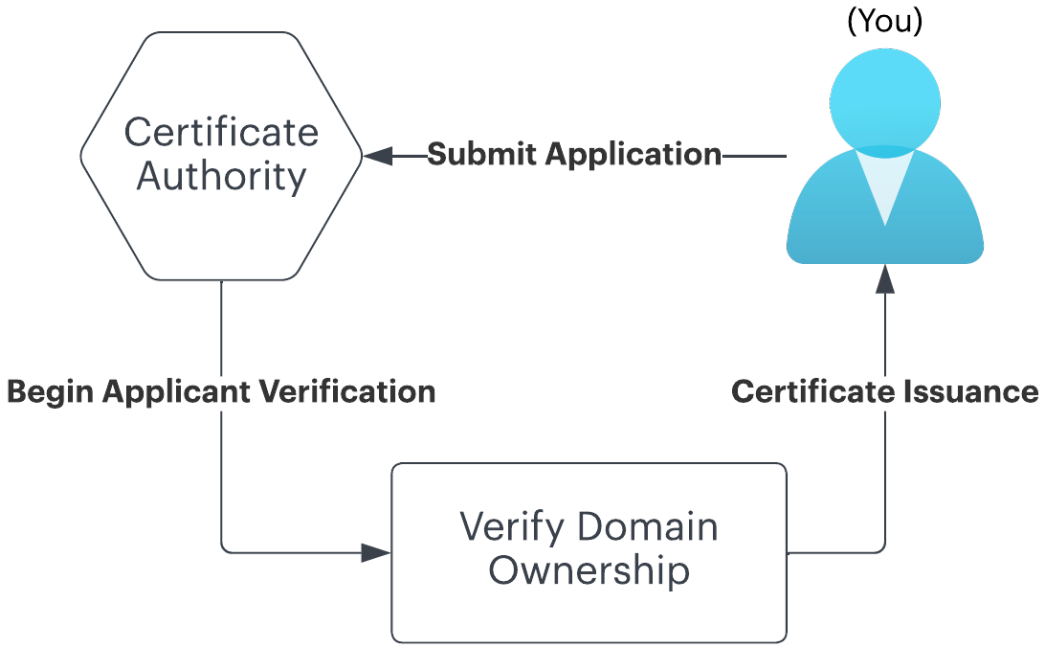
Validation Type	Description	Verification Level	Potential for Misuse
Domain Validation (DV)	Proof of domain control	**Low** Only domain is validated	**High** Easy for attackers to obtain and misuse
Organization Validation (OV)	Proof of legal and physical existence of the organization	**Moderate** Organization details are validated	**Moderate** More verification steps reduce risk
Extended Validation (EV)	Rigorous validation of the organization, including physical presence and legal existence	**High** Thorough validation increases trust level	**Low** Rigorous process and higher costs deter misuse

Review: Identity Verification Requirements

Extended Validation Process



Domain Validation Process



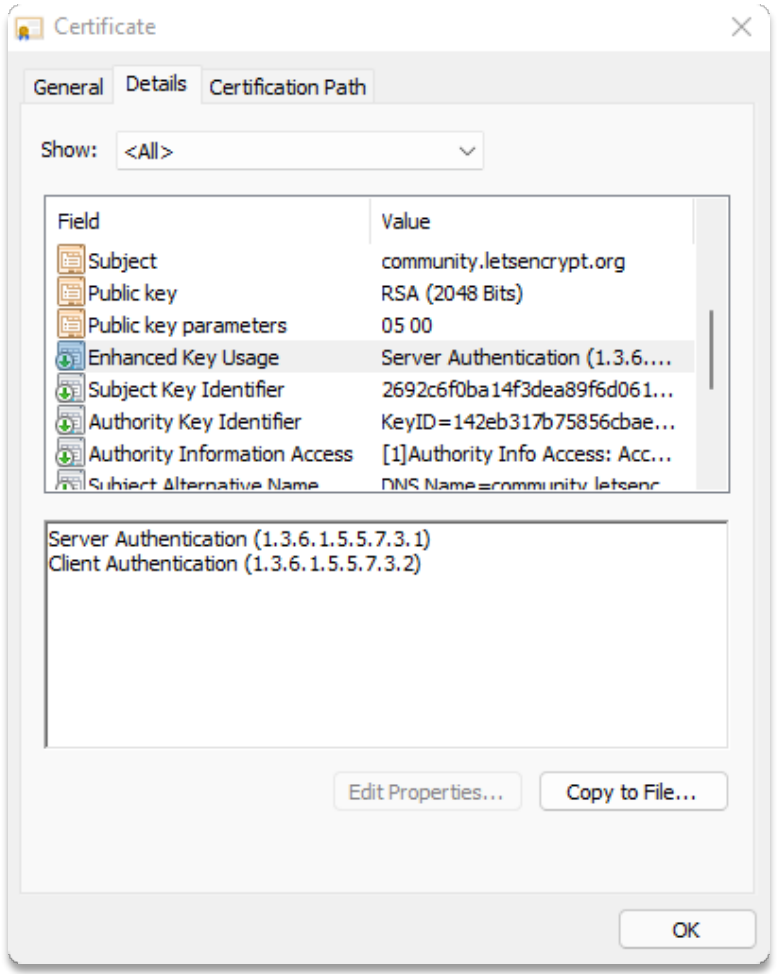
Abusing the Lack of EKU Validation

What safeguards exist to prevent DV certificate misuse in contexts requiring higher verification?

What *Defines* a Certificate's Purpose?

What differentiates a Code Signing Certificate from an SSL Certificate? Introducing the **Extended Key Usage** (EKU) field!

- EKUs play a pivotal role in defining what actions a certificate can authenticate, like server authentication, client authentication, code signing, etc.
- EKUs are part of the certificate extensions, expanding the functionality of certificates by defining their purpose.



What *Verifies* a Certificate's Purpose?

Restrictions Imposed by Signing Tools

- When you sign an executable, email, or other data, your client can restrict certificates by their EKU.
- This is no more than a client-side sanity check.

Restrictions Imposed by Signature Validation Implementations

- When the recipient's client receives the data that has been digitally signed, it can verify that the certificate has an appropriate EKU field.
- This is the step that matters.

Preparing Our Attack

What do we need to be able to identify EKU validation vulnerabilities?

Identifying Our Criteria

MSRC wants to protect the entire ecosystem, not just first-party implementations. How do we define a criteria with this goal in mind?

- An “attacker” certificate was generated, meeting the bare minimum standard of Domain Validation.
- What about our scope?
  - Code Signing of Executable Formats **handled by the OS**
    - .exe, .dll, .msi, .js, etc.
  - Code Signing of Executable Formats **handled by External Software**
    - Macros in Word Documents, NuGet Packages, ClickOnce, etc.
  - Miscellaneous Formats that support digital signatures
    - PDF, DOC(X), etc.

Modifying Signing Tools to Remove EKU Validation

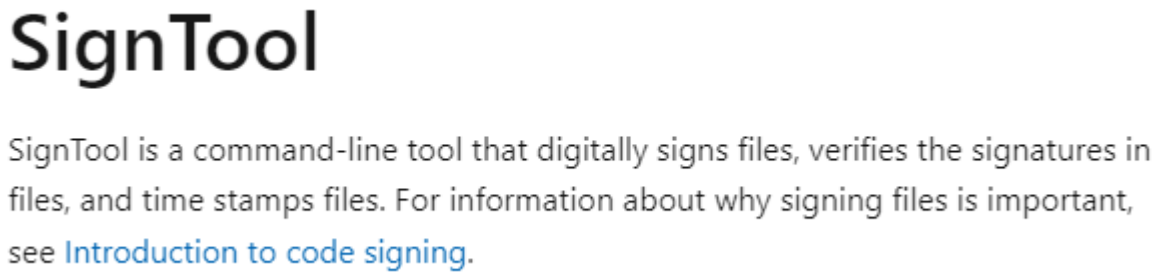
In the next few slides, we'll review the signing tools used for the formats in our scope and modify them to remove any EKU validation.

- Most of the utilities we found to sign executable formats verify that your certificate has the Code Signing EKU.
- We need to patch these executables to skip this step.
- This is only getting rid of the client-side verification to create samples to test various implementations with.

Example: Reversing & Modifying SignTool

One of the most popular signing tools on Windows is **signtool**.

- Supports over ~25 unique file extensions.
  - .exe, .cab, .cat, .dll, .cpl, .js, .msi, .msix*, etc.
- Commonly used for Authenticode formats.
- Included with the Windows SDK.



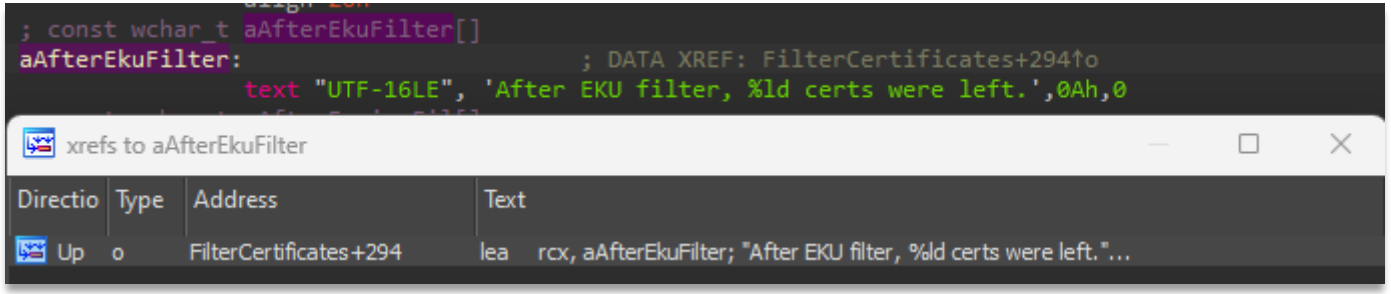
Example: Reversing & Modifying SignTool

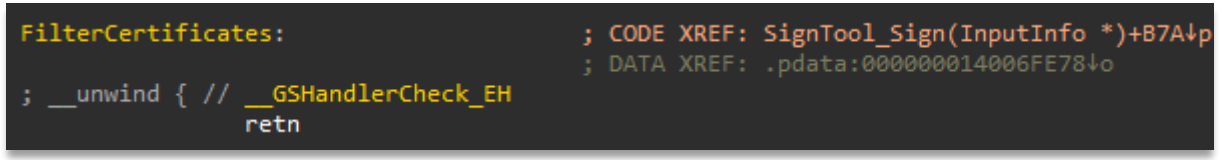
What happens if we try to sign an executable with an SSL certificate?

```
The following certificates were considered:
    Issued to: test.billdemirkapi.me
    Issued by: R3
    Expires:   Thu Jun 01 11:40:42 2023
    SHA1 hash: A8E3961548698A50E273E15D0ADB69329733FF2C

After EKU filter, 0 certs were left.
After expiry filter, 0 certs were left.
After Private Key filter, 0 certs were left.
SignTool Error: No certificates were found that met all the given criteria.
```

Example: Reversing & Modifying SignTool

- I used IDA Pro with public symbols to search for related strings.
- Looking for “After EKU Filter…” quickly revealed the function responsible for filtering certificates.
- How do we prevent our SSL certificate from being filtered? Add a simple “ret” instruction patch to the beginning of the function!

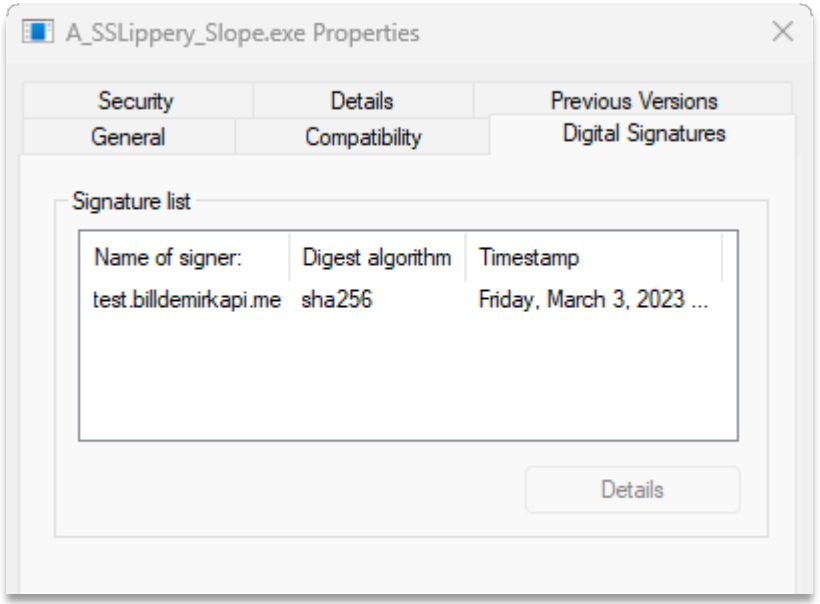




Example: Reversing & Modifying SignTool

The next time I signed the executable, signtool had no complaints with an SSL certificate!

```
The following certificates were considered:
    Issued to: test.billdemirkapi.me
    Issued by: R3
    Expires:   Thu Jun 01 11:40:42 2023
    SHA1 hash: A8E3961548698A50E273E15D0ADB69329733FF2C

The following certificate was selected:
    Issued to: test.billdemirkapi.me
    Issued by: R3
    Expires:   Thu Jun 01 11:40:42 2023
    SHA1 hash: A8E3961548698A50E273E15D0ADB69329733FF2C


The following additional certificates will be attached:
    Issued to: R3
    Issued by: ISRG Root X1
    Expires:   Mon Sep 15 11:00:00 2025
    SHA1 hash: A053375BFE84E8B748782C7CEE15827A6AF5A405

Done Adding Additional Store
Successfully signed: test.exe

Number of files successfully Signed: 1
Number of warnings: 0
Number of errors: 0
```



Example: Reversing & Modifying MAGE

- Although signtool handles many Authenticode use cases, certain non-binary formats such as .application and .manifest must be signed using **MAGE**.

Mage.exe (Manifest Generation and Editing Tool)

The Manifest Generation and Editing Tool (*Mage.exe*) is a command-line tool that supports the creation and editing of application and deployment manifests. As a command-line tool, *Mage.exe* can be run from both batch scripts and other Windows-based applications, including ASP.NET applications.

Example: Reversing & Modifying MAGE

- Like signtool, MAGE doesn't allow you to sign manifests with certificates that lack the Code Signing EKU.
- Unlike signtool, MAGE is not a C++ application- it's written in C#.
- How do we reverse engineer its EKU check?

```
C:\Users\[redacted]\[redacted]\test.billdemirkapi.me>mage
-sign manifest/test.application -CertFile test.billdemirkapi.me.pfx
This certificate cannot be used for signing - "test.billdemirkapi.me.pfx"
```

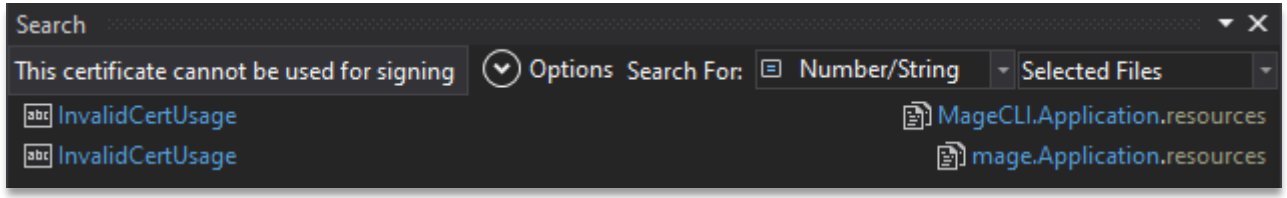
dnSpy - Latest release

dnSpy is a debugger and .NET assembly editor. You can use it to edit and debug assemblies even if you don't have any source code available. Main features:

- Debug .NET and Unity assemblies
- Edit .NET and Unity assemblies

Example: Reversing & Modifying MAGE

- Searching for the error led to the responsible function.
- **CanSignWith** verifies the presence of the Code Signing EKU.
- dnSpy allows us to edit this method. Like signtool, we can add a simple return statement.




```
static bool CanSignWith(X509Certificate2 cert)
{
    bool result = true;
    foreach (X509Extension x509Extension in cert.Extensions)
    {
        bool? flag = null;
        if (string.Compare(x509Extension.Oid.Value, Certificate.szOID_KEY_USAGE, true,
          CultureInfo.InvariantCulture) == 0)
        {
            X509KeyUsageExtension x509KeyUsageExtension = new X509KeyUsageExtension();
            x509KeyUsageExtension.CopyFrom(x509Extension);
            if ((x509KeyUsageExtension.KeyUsages & X509KeyUsageFlags.DigitalSignature) !=
              X509KeyUsageFlags.None)
            {
                foreach (X509Extension extension in cert.Extensions)
                {
                    flag = Certificate.CodeSignEnhancedKeyPresent(extension);
                    if (flag != null)
                    {
                        result = flag.Value;
                        break;
                    }
                }
```

```
static bool CanSignWith(X509Certificate2 cert)
{
    return true;
}
```

Example: Reversing & Modifying MAGE

- After we replaced the **CanSignWith** method, we can sign manifests using our SSL certificate!

```
C:\Users\[redacted]\[redacted]\test.billdemirkapi.me>mage
-sign manifest/test.application -CertFile test.billdemirkapi.me.pfx
test.application successfully signed
```

Exploring Authenticode Implementations

Let’s test our samples against the real-world!

Background: Attacking Authenticode on Windows

- WinVerifyTrust is a complex API that performs various “trust verification action[s] on a specified object”.
- It abstracts the job of signature validation to various “subject interface packages” (SIPs) and “trust providers”.
- SIPs are responsible for the format-specific verification of digital signatures, as each specification will store them in different locations.
- This extensible architecture supports numerous formats:
  - .exe, .cab, .cat, .ps1*, .msi, .js, etc.

Read more: Subverting Trust in Windows by Matt Graeber

Background: Attacking Authenticode on Windows

- Trust providers do not care about the SIP you use- they are designed to perform specific “trust verification actions“.
- Each provider has a different purpose. A few common ones include…
  - **WINTRUST_ACTION_GENERIC_CERT_VERIFY** – Verify a certificate chain only.
  - **WINTRUST_ACTION_GENERIC_CHAIN_VERIFY** – Verify certificate chains created from any object type.
  - **WINTRUST_ACTION_GENERIC_VERIFY_V2** – Verify a file or object using the Authenticode policy provider.
- The last provider listed is most used for verifying an Authenticode signature’s authenticity.

Attacking Authenticode on Windows

- I compiled a quick test application that uses WinVerifyTrust to check the signature of an executable.
- (~~Un~~)Fortunately, it didn't work.
- 0x800b0110 represents the error "CERT_E_WRONG_USAGE".
  - "The certificate is not valid for the requested usage".



Attacking Authenticode on Windows

- The generic trust provider used for most extensions mandates the Code Signing EKU.
- (~~Un~~)Fortunately, this meant most executable formats are protected by default.

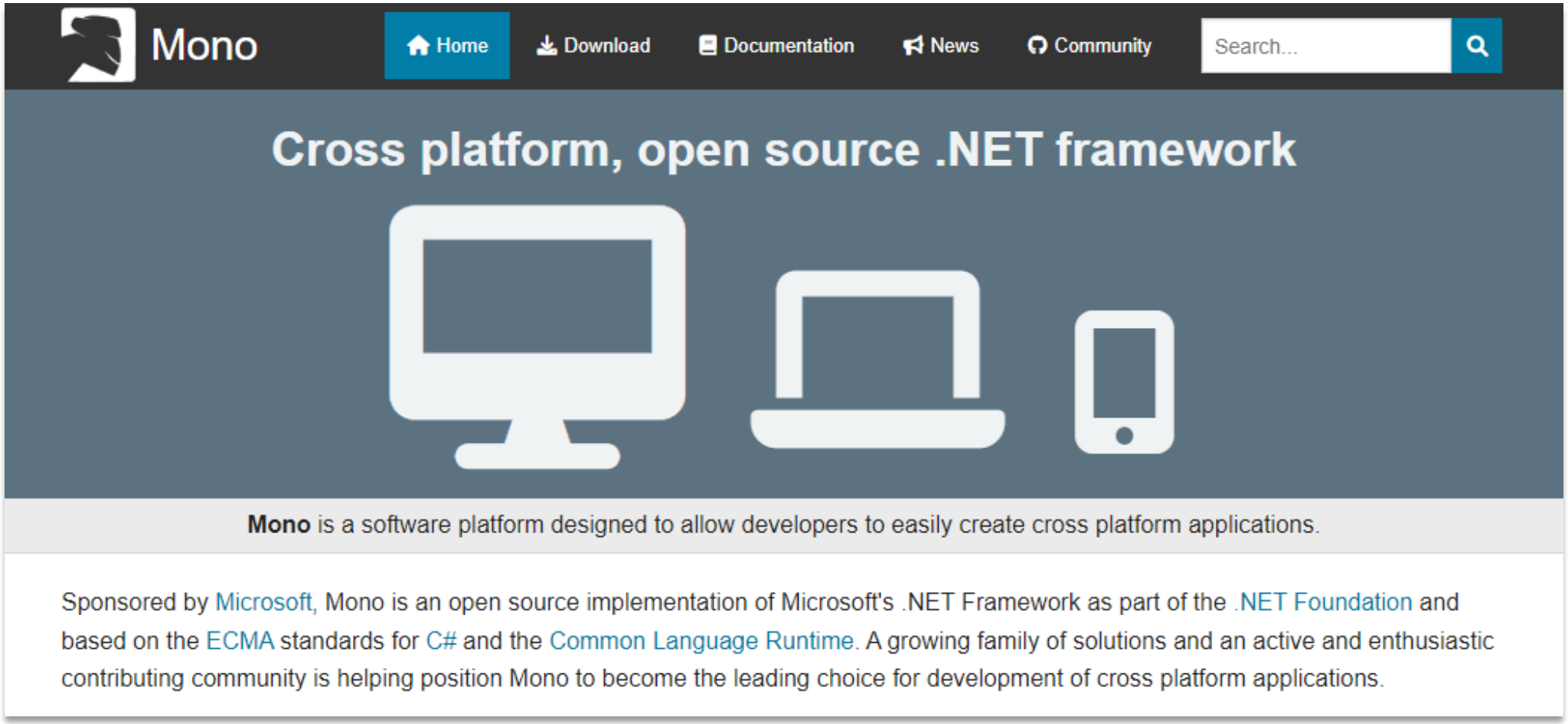
6. **FinalPolicy:**
    a. This is the function where the majority of trust decisions are made. At this point, the signature and certificate chain has been decoded, parsed and supplied to this implementing function.
    b. What components of the signature, certificate chain, and certificate store are validated vary depending upon the trust provider. Here is small list of some of the checks that occur when the `WINTRUST_ACTION_GENERIC_VERIFY_V2` trust provider is used (implemented as wintrust!SoftPubAuthenticode):
        i. Verify that the file was signed with a certificate designated for code signing as indicated by an enhanced key usage (EKU) of "1.3.6.1.5.5.7.3.3"
        ii. Check if the certificate is expired and hasn't been timestamped.
        iii. Check if the certificate has been revoked.
        iv. Validate that the file was not signed using a hash algorithm that has been designated as "weak."
        v. If the file was signed with a certificate designated for "Windows System Component Verification" (EKU - 1.3.6.1.4.1.311.10.3.6), validate that the signing certificate chains to a fixed set of trusted Microsoft root certificates.

Attacking Third-Party Authenticode Implementations

- Although the Windows implementation of WinVerifyTrust is secure, I was curious about other libraries that validated the authenticity of Authenticode applications.
- For example, there are a significant number of workloads that may need to verify an executable's signature on other operating systems.
- Unfortunately, several major external Authenticode implementations were vulnerable to this attack.

First-Party Example: Vulnerability in the Mono Project

- Introducing the Mono Project!



First-Party Example: Vulnerability in the Mono Project

- The Mono Project supports Authenticode signing and verification with the Mono.Security.Authenticode namespace.

Mono.Security.Authenticode Namespace

Classes

Type	Reason
AuthenticodeBase	Common base class for both Authenticode Formatter (signature generation) and Deformatter (signature validation).
AuthenticodeDeformatter	Authenticode(r) compatible code signing verification.
AuthenticodeFormatter	Authenticode(r) compatible code signing generation.

Authenticode(r) Code Signing

Does Mono support Authenticode(r) signatures ?

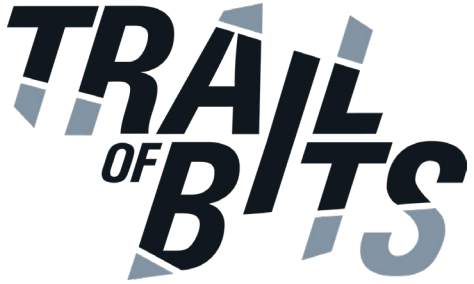
Yes. Mono supports Authenticode signatures for **assemblies**. As assemblies are **PE** files this also means that Mono support signing any kind of PE file, e.g. EXE, DLL, OCX, SCR ...

First-Party Example: Vulnerability in the Mono Project

- Due to a lack of EKU validation, Mono's **AuthenticodeDeformatter** class is vulnerable to the attack.
- This issue impacts any application that uses the class to verify untrusted executables.

```
[redacted]:/mnt/c/Users/[redacted]test.billdemirkapi.me$ chktrust helloworld.exe
Mono CheckTrust - version 6.8.0.105
Verify if an PE executable has a valid Authenticode(tm) signature
Copyright 2002, 2003 Motus Technologies. Copyright 2004-2008 Novell. BSD licensed.

SUCCESS: helloworld.exe signature is valid
and can be traced back to a trusted root!
```

Third-Party Example: Vulnerability in µthenticode



uthenticode

Tests passing

uthenticode (stylized as *µthenticode*) is a small cross-platform library for verifying Authenticode digital signatures.

What?

Authenticode is Microsoft's code signing technology, designed to allow signing and verification of programs.

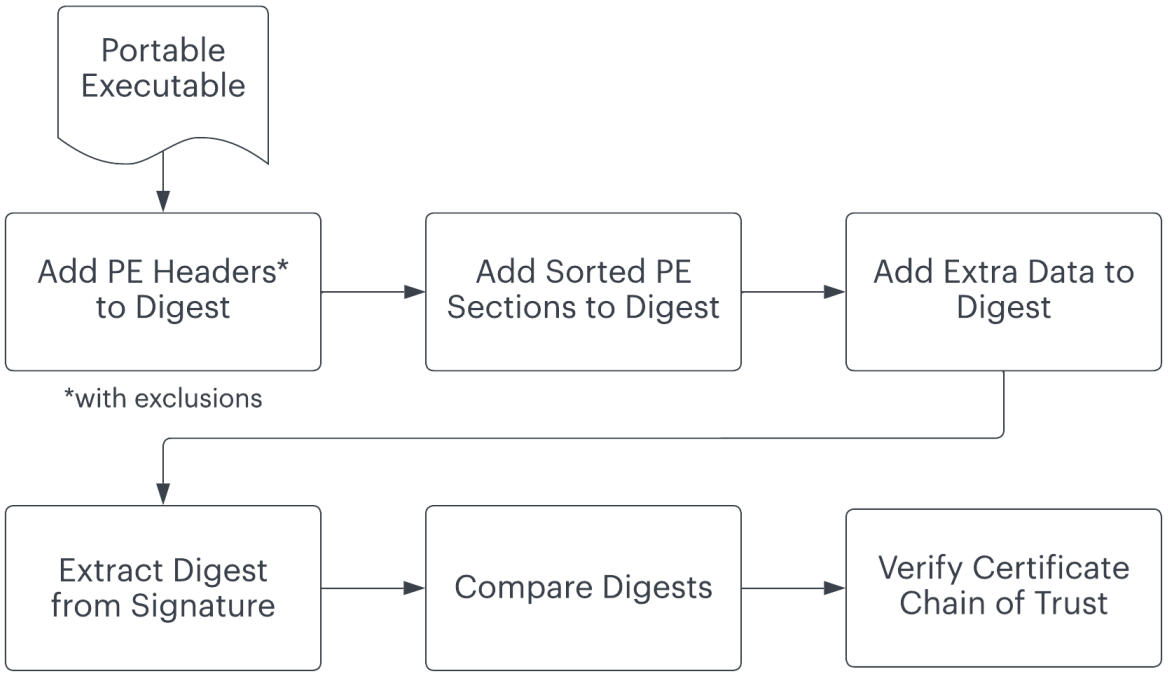
µthenticode is a cross-platform reimplementation of the verification side of Authenticode. It doesn't attempt to provide the signing side.

Third-Party Example: Vulnerability in µthenticode

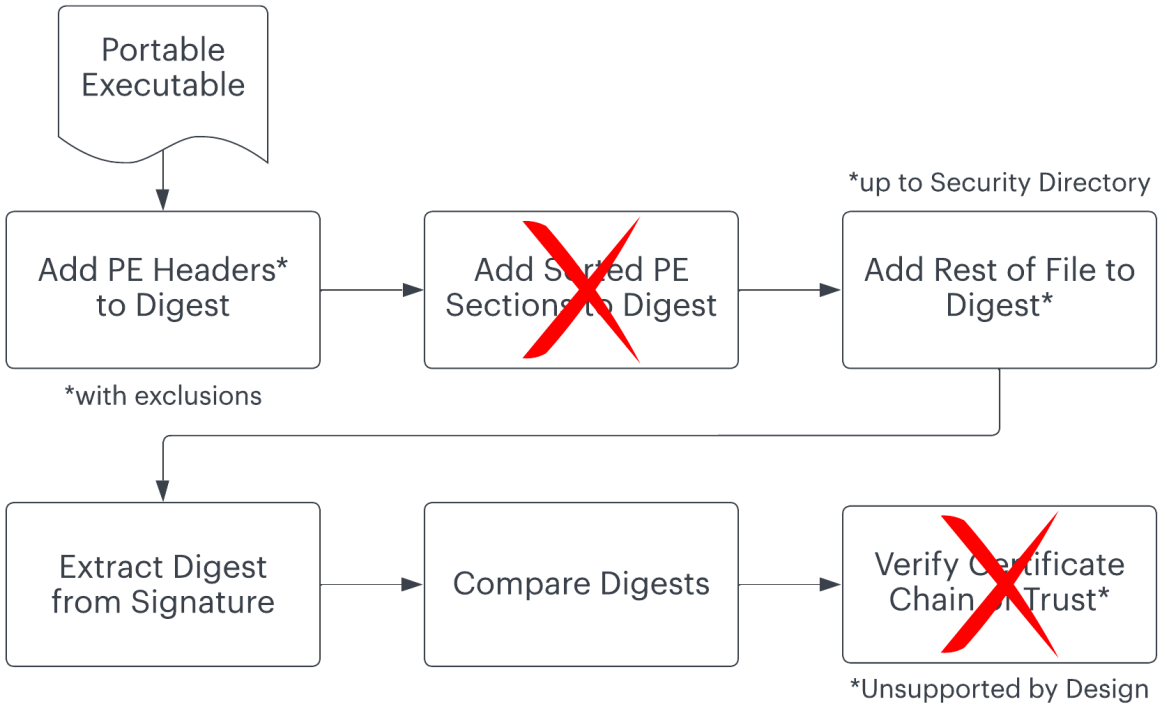
```
                                    \uthenticode\build>src\svcli\Debug\svcli.exe helloworld.exe
This PE is verified!

helloworld.exe has 1 certificate entries

Calculated checksums:
   MD5: 7C2963535A90976BD1A40B5A2F1AFFC7
  SHA1: DB936C01A55E53DBB55392204A5B546D60FD6C41
SHA256: 0F7EC8D14CCDE438174CD9F25EBE9DDCCED47576A5049A459B4F610333ED0F90

SignedData entry:
        Embedded checksum: 0F7EC8D14CCDE438174CD9F25EBE9DDCCED47576A5049A459B4F610333ED0F90

        Signers:
                Subject: /CN=test.billdemirkapi.me
                Issuer: /C=US/O=Let's Encrypt/CN=R3
                Serial: 03502772348722773B93EDC07DB49149E207

        Certificates:
                Subject: /C=US/O=Let's Encrypt/CN=R3
                Issuer: /C=US/O=Internet Security Research Group/CN=ISRG Root X1
                Serial: 912B084ACF0C18A753F6D62E25A75F5A

                Subject: /CN=test.billdemirkapi.me
                Issuer: /C=US/O=Let's Encrypt/CN=R3
                Serial: 03502772348722773B93EDC07DB49149E207

        This SignedData is valid!
```

Bonus Example: Vulnerability in µthenticode

Typical Authenticode Flow



µthenticode's Flow



Surely there are no unintended consequences from deviating from the specification… right?

Bonus Example: Vulnerability in µthenticode

- The problem? An attacker can manipulate a PE Section *without* breaking the signature by embedding the security directory in it.

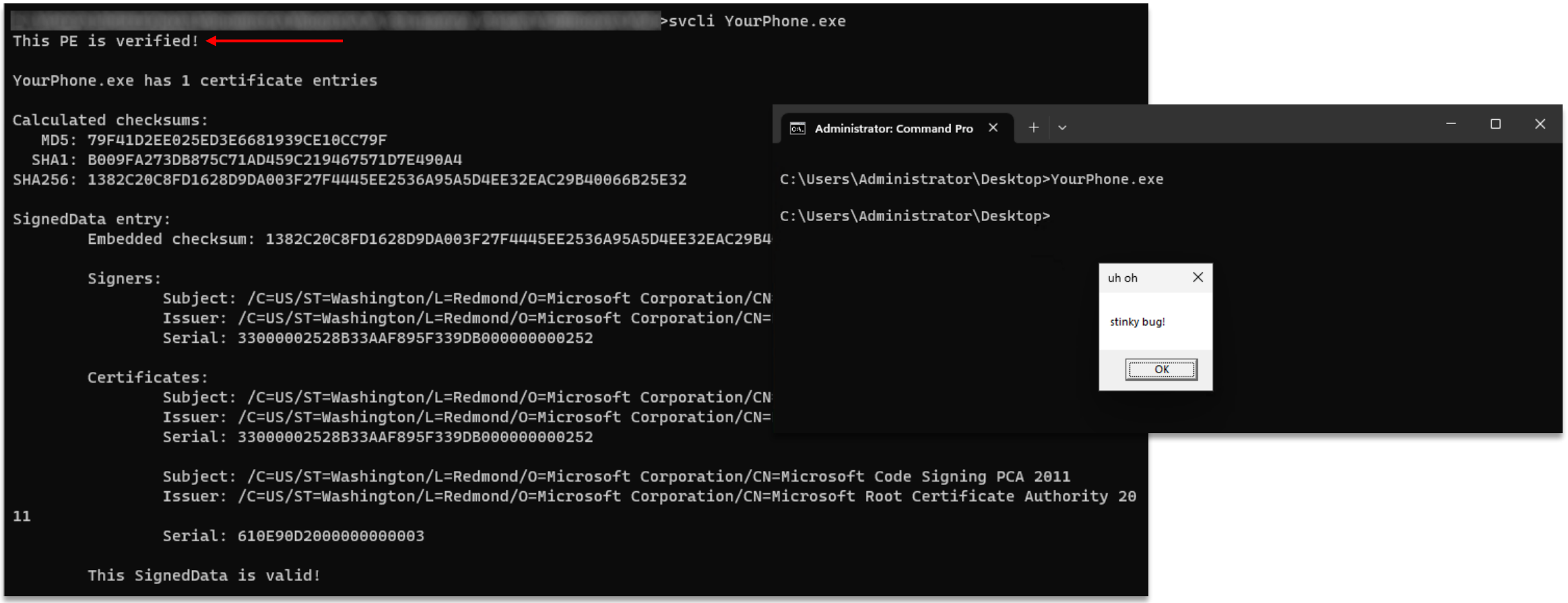
Typical Portable Executable

DOS Header
PE Header
Optional Header
Sections Table
.text Section (Code)
Other PE Sections (e.g., Data)
Security Directory

Corrupted Portable Executable

DOS Header	
PE Header	
Optional Header	
Sections Table	
.text Section (Code) Start	
Security Directory (Malicious Code)	(Excluded)
.text Section (Code) End	
Other PE Sections (e.g., Data)	

Bonus Example: Vulnerability in µthenticode



Exploring ClickOnce on Windows

Enough about regular executables; What about other signing implementations?

What is ClickOnce and Why Does It Matter?

- ClickOnce is a Microsoft technology that enables user-friendly application deployment on Windows, focusing on simplicity and security™.

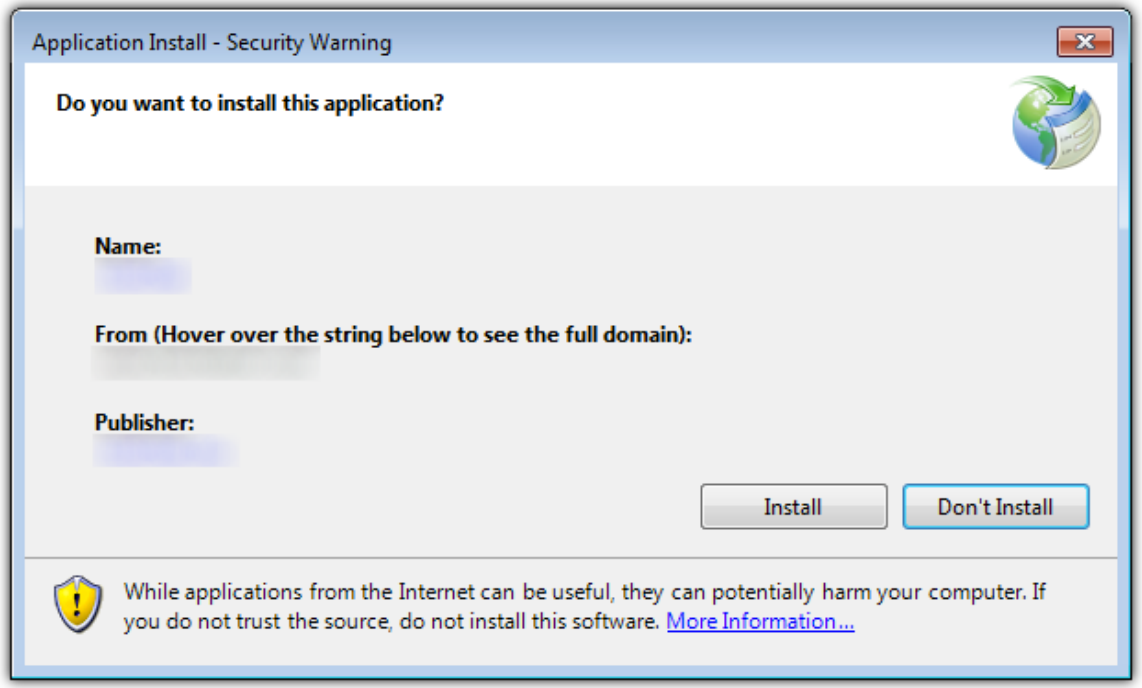
ClickOnce security and deployment

ClickOnce is a deployment technology that enables you to create self-updating Windows-based applications that can be installed and run with minimal user interaction. Visual Studio provides full support for publishing and updating applications deployed with ClickOnce technology if you have developed your projects with Visual Basic and Visual C#. For information about deploying Visual C++ applications, see ClickOnce Deployment for Visual C++ Applications.

How Does It Work?

ClickOnce deployments involve multiple components:

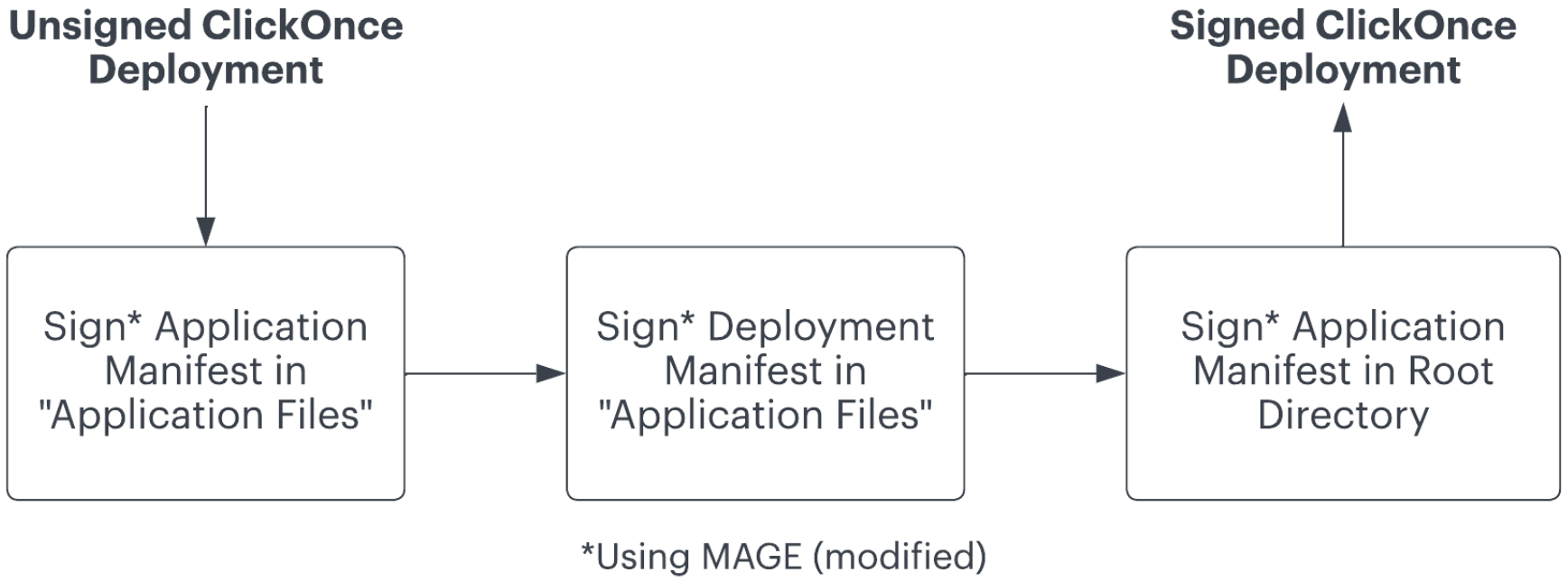
- Application files
- A deployment manifest
- An optional application manifest



Due to its flexibility, you can install a ClickOnce application by clicking a link on a web page, through a file share, or from legacy media.

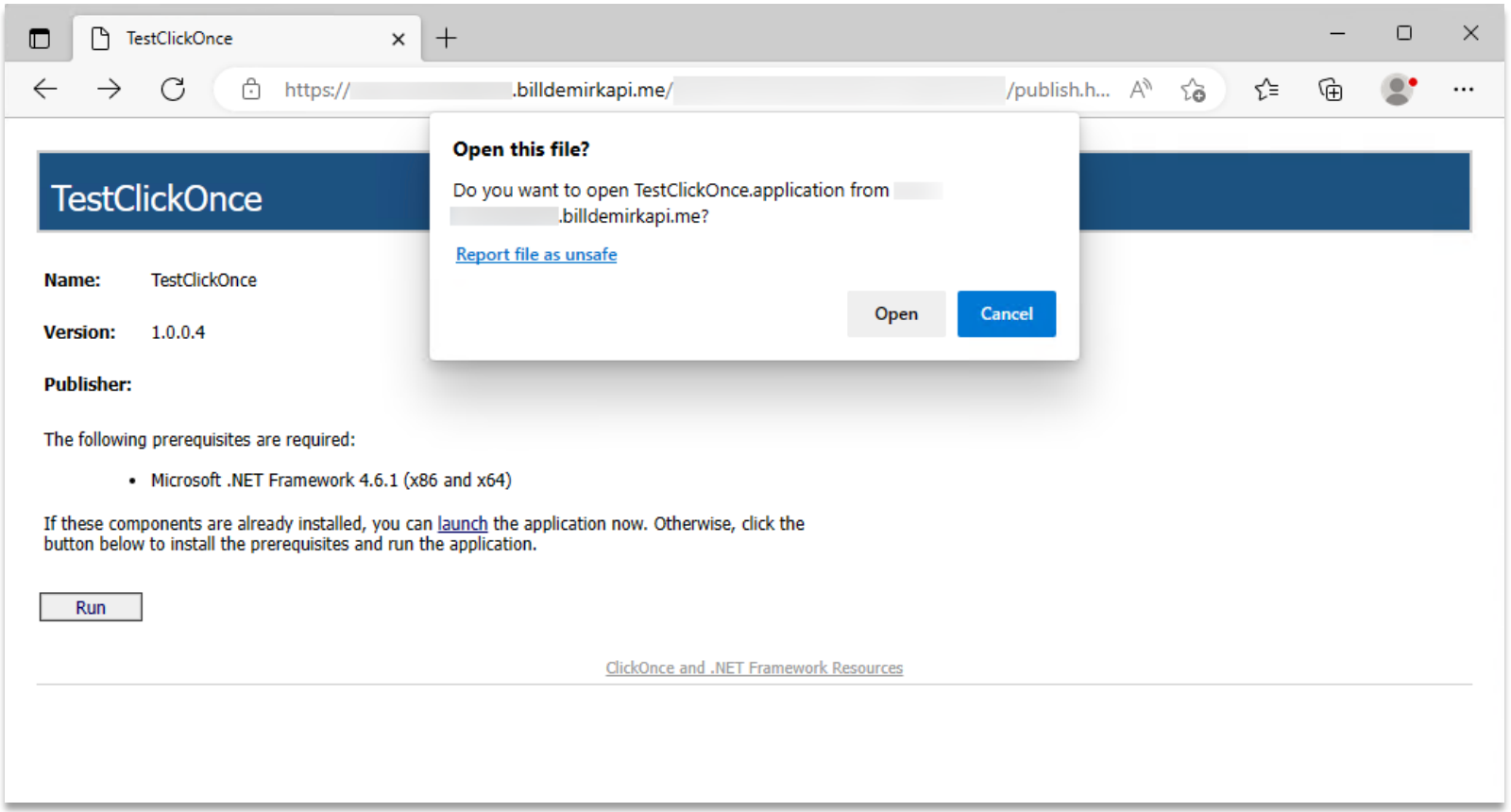
First-Party Example: Vulnerability in ClickOnce

- ClickOnce applications can be signed automatically with Visual Studio or manually with MAGE.
- What does that look like?



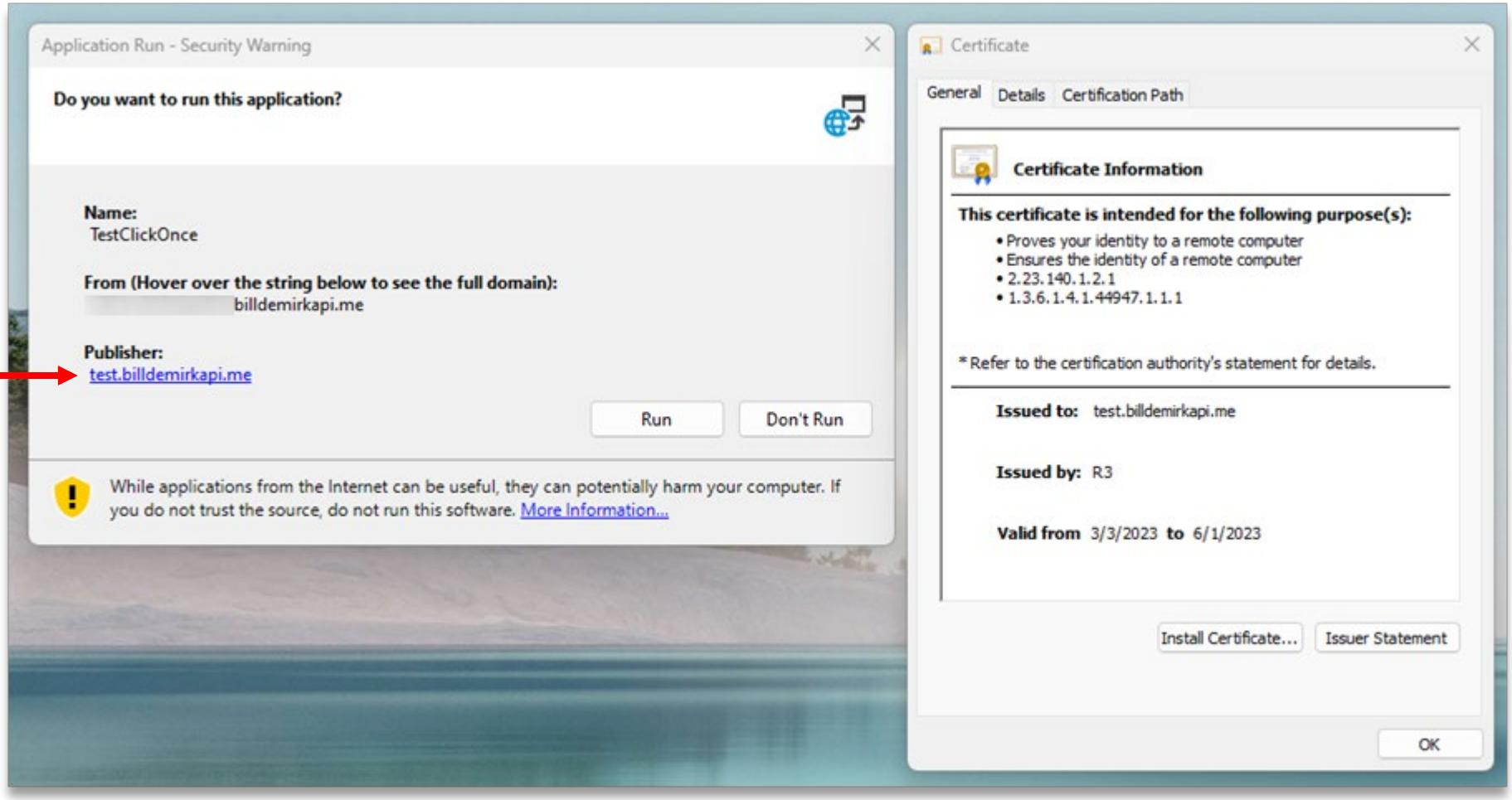
First-Party Example: Vulnerability in ClickOnce

- Once the deployment was signed, it was uploaded to a web server for testing.



First-Party Example: Vulnerability in ClickOnce

- Unfortunately, ClickOnce is impacted by the attack.



Exploring Everything Else & Future Work

What about other uses of digital signatures we didn’t cover?

Related Work

In 2014, researchers from the University of Texas and University of California released "*Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations*".

- As the name suggests, the researchers generated mutated certificates with different "combinations of extensions and constraints".
  - The idea was to look for inconsistencies between implementations.
- The paper exclusively focuses on testing the SSL/TLS implementations in OpenSSL, NSS, GnuTLS, CyaSSL, PolarSSL, etc.

Related Work

- Of relevance to this talk, the researchers found several libraries that failed to validate the appropriate EKU.
- Instead of malformed certificates for digital signatures in executables, they looked at abusing them in the browser.

Server certificate not authorized for use in SSL/TLS handshake. PolarSSL, GnuTLS, CyaSSL, and MatrixSSL do not check the key usage extension in leaf certificates. This is a serious security vulnerability. For example, if an attacker compromises some company's code signing certificate, which is only intended for authenticating code, he will be able to impersonate that company's network and Web servers to any application based on the above SSL/TLS libraries, vastly increasing the impact of the attack.

Server certificate not authorized for server authentication. PolarSSL, gnuTLS, CyaSSL and MatrixSSL do not check the extended key usage extension. Given a certificate with key usage that allows all operations and extended key usage that only allows it to be used for TLS *client* authentication (or any purpose other than server authentication), these libraries accept the certificate for server authentication.

Read more: https://github.com/sumanj/frankencert

Example: OpenSSL, EKU Validation, and How It Can Be Disabled

We additionally verified that common libraries like OpenSSL validate EKUs appropriately.

- If you use these libraries for solely verifying the authenticity of a certificate, without specifying a purpose (and no default), you can still be exposed to an EKU attack.
- In OpenSSL, you can specify a context for X509 verification, which can include EKU defaults.

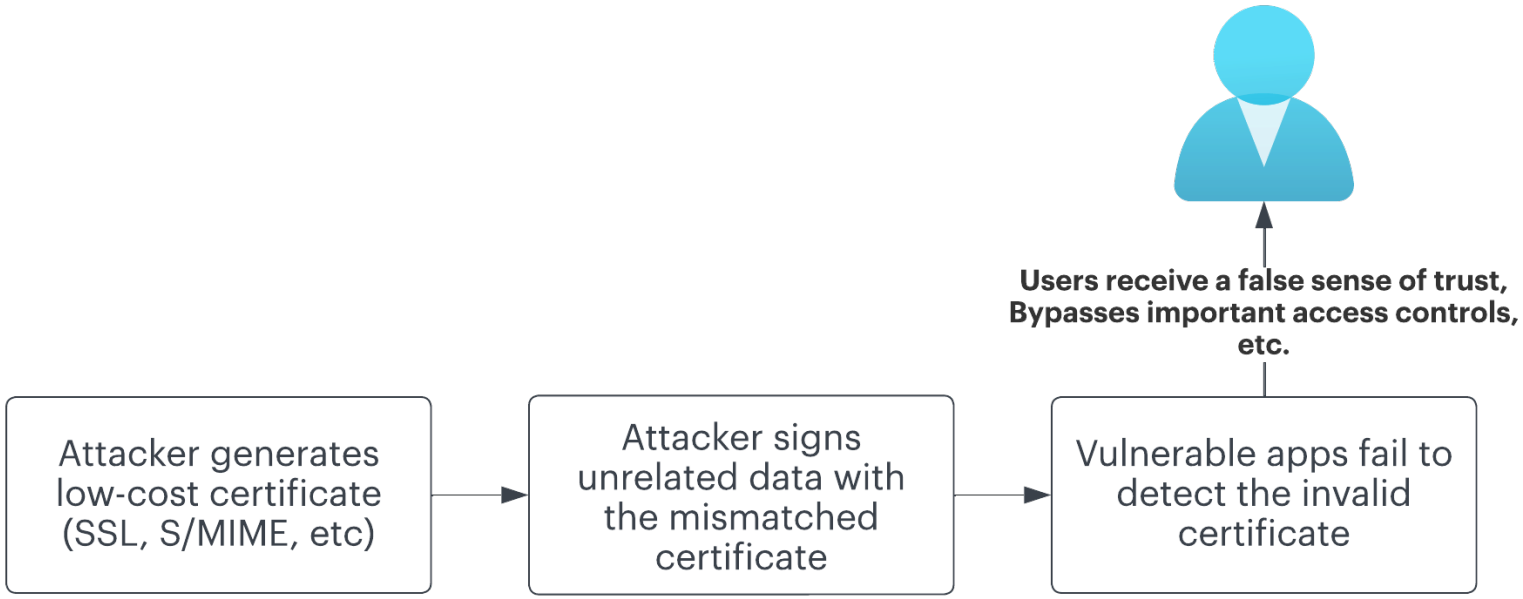
```
/*
 * We need to inherit the verify parameters. These can be determined by
 * the context: if its a server it will verify SSL client certificates or
 * vice versa.
 */

    X509_STORE_CTX_set_default(ctx, s->server ? "ssl_client" : "ssl_server");
```

Conclusion

Time to review takeaways and techniques.

Reviewing the Attack

- In this project, we discovered numerous implementations of digital signatures that failed to validate the Extended Key Usage field.
- This would allow an attacker to abuse certificates with a substantially lower bar for identity verification in important contexts, like code signing.

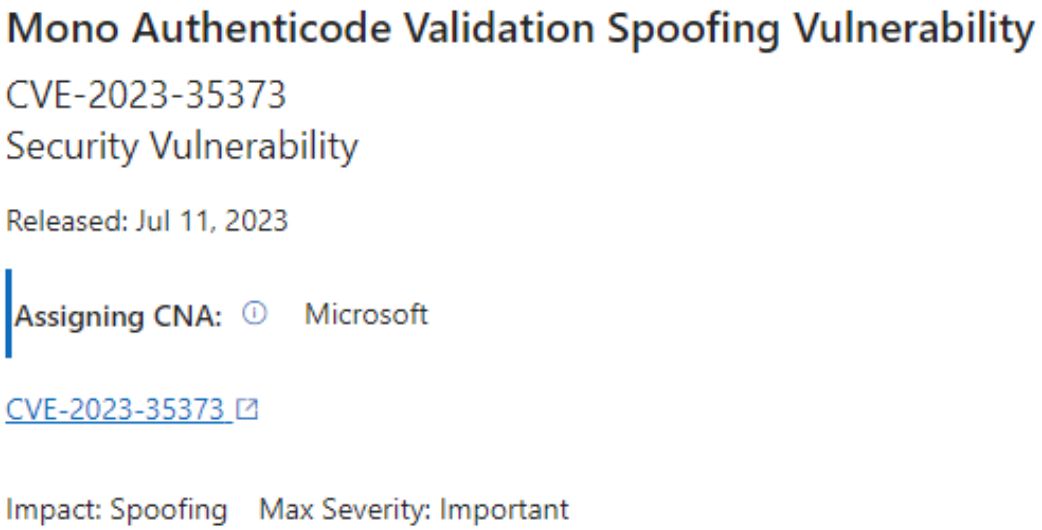


How to Protect Your Implementation?

- **Point-Fix**: Always validate EKUs in your application to ensure that certificates are used for their intended purpose.
  - Use libraries that properly implement EKU checks.
- Trust, but verify key principles in your application's design.
  - All vulnerable implementations we found were "supposed to" check for EKUs. No one verified that they did.
- Implement regular security testing, especially in components using cryptography for crucial features.
  - Your implementation is much more likely to have a vulnerability than a cryptographic algorithm.

What Is MSRC Doing to Protect Customers?

- We've released patches for all first-party issues.
- The third-party vulnerabilities we discussed have also been fixed.
- We're continuing to work with impacted third-party vendors to address their implementations.
- We continue to explore digital signature implementations.



Questions

Thanks for coming to my talk! Now is the time for questions.

Twitter: @BillDemirkapi
Personal Blog: https://billdemirkapi.me/